

DÉVELOPPEMENT N-TIERS J2EE

<http://www.larbiquezouli.com>

ou

http://fac-sciences.univ-batna.dz/cs/enseignants/guezouli_larbi_site/index.html

Présenté par Dr Larbi GUEZOULI

DÉVELOPPEMENT N-TIERS J2EE

- Chapitre I: Introduction [1 séance]
 - Evolution des architectures client-serveur
 - Objets répartis et architecture client-serveur
 - Systèmes transactionnels
 - Serveurs d'applications à base de composants
- Chapitre II: Architecture J2EE [2 séances]
 - Conteneurs J2EE
 - Vue d'ensemble d'application J2EE
- Chapitre III: Couche métier et persistance [3 séances]
 - Serveur EJB
 - Modèle de composants EJB
- Chapitre IV: Couche présentation [3 séances]
 - Les servlets
 - La technologie JSP
 - Communication entre servlet et vue
 - La session en J2EE

Mode d'évaluation :

Examen final : coef. 2

Contrôle continu (TP) : coef. 1

2

CHAPITRE I INTRODUCTION

PLAN

1. Evolution des architectures client-serveur
2. Objets répartis et architecture client-serveur
3. Systèmes transactionnels
4. Serveurs d'applications à base de composants

3

CHAPITRE I INTRODUCTION

1. Evolution des architectures client-serveur

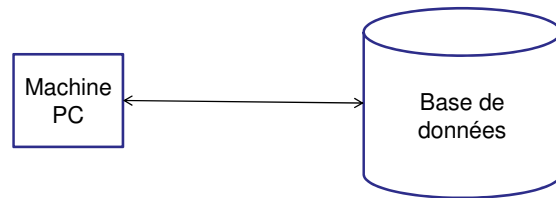
- Architecture des systèmes centraux
 - Tout est centralisé, pas de réseau.
- Architecture client-serveur
 - Un serveur et des milliers de clients
- Navigateurs Web
 - Information sur serveur, consulté par des millions de clients.

4

CHAPITRE I

INTRODUCTION

- **Architecture 1 tier (des systèmes centraux)**

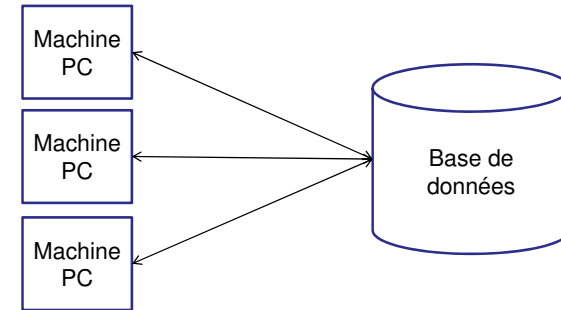


5

CHAPITRE I

INTRODUCTION

- **Architecture 2 tiers (client-serveur)**

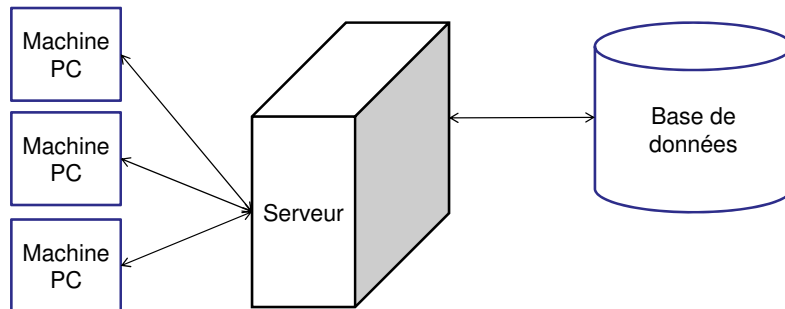


6

CHAPITRE I

INTRODUCTION

- **Architecture 3 tiers (RCP)**

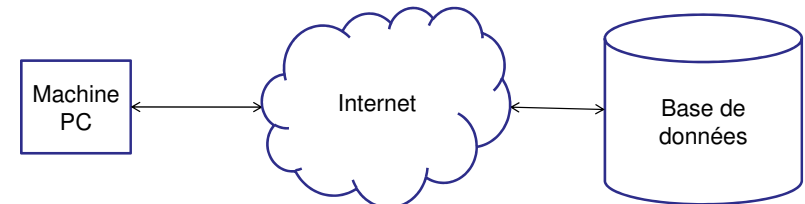


7

CHAPITRE I

INTRODUCTION

- **Architecture 3 tiers (Internet)**



8

CHAPITRE I

INTRODUCTION

2. Objets répartis et architecture client-serveur

Les objets répartis ou distribués permettent de distribuer une application.

Un objet distribué peut être appelé par:

- un client qui appartient au processus contenant l'objet;
- un client de l'extérieur du processus contenant l'objet;
- un client distant sur le réseau.

9

CHAPITRE I

INTRODUCTION

3. Systèmes transactionnels

Sont des systèmes qui assurent la réalisation de transactions.

Une transaction est définie comme une unité de code **indivisible**.

Exp.

Dans un système de e-commerce, une transaction de paiement en ligne doit débiter le compte de l'acheteur, et créditer le compte du vendeur.

Si une opération de ces deux opérations n'est pas faite, le tout sera annulé.

10

CHAPITRE I

INTRODUCTION

4. Serveurs d'applications à base de composants

Solutions commerciales comme:

- Weblogic server
- WebSphere Application Server
- VisualAge for Java
- iPortal Application Server
- iPlanet Application Server
- Oracle9iAS
- eXtend Application Server
- EAServer

11

CHAPITRE I

INTRODUCTION

4. Serveurs d'applications à base de composants (suite)

Solutions open-source comme:

- JBoss
- Enhydra
- Apache Projects
- Xerces
- Xalan
- SOAP
- Jakarta
- Tomcat
- ...

12

CHAPITRE II

ARCHITECTURE J2EE

PLAN

1. Conteneurs J2EE
 1. Contexte
 2. Les Conteneurs (livrables)
 1. La livraison EAR (Entreprise ARchive)
 2. La livraison Web WAR (Web ARchive)
 3. La livraison EJB (Entreprise JavaBean) (.JAR)
 4. La livraison Client (Module Client) (.JAR)
 5. La livraison Ressources (Module Connector) (.RAR)
2. Vue d'ensemble d'application J2EE
 1. Création
 2. Assemblage
 3. Déploiement

13

CHAPITRE II

ARCHITECTURE J2EE

1. Conteneurs J2EE

1.1. Contexte

Un projet J2EE est développé par plusieurs équipes. Chaque équipe développe une partie appelée « Module » comme indiqué sur la figure suivante:

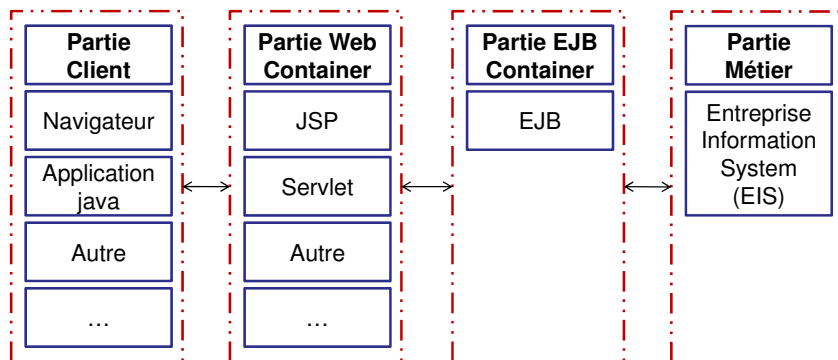
14

CHAPITRE II

ARCHITECTURE J2EE

1.1. Contexte (suite) :

Architecture n-tiers

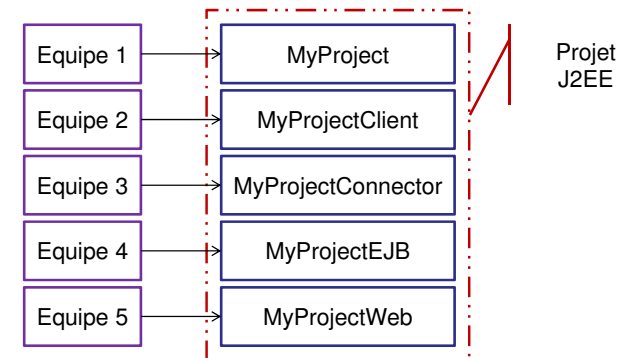


15

CHAPITRE II

ARCHITECTURE J2EE

1.1. Contexte (suite)



16

CHAPITRE II

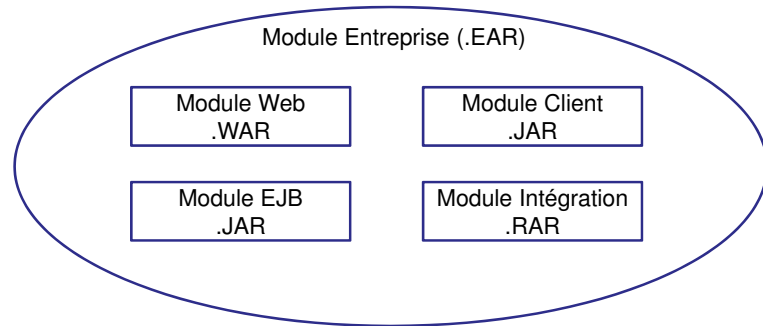
ARCHITECTURE J2EE

1.2. Les Conteneurs (livrables)

1.2.1. La livraison EAR (Entreprise ARchive)

C'est la livraison qui englobe tous les modules du projet J2EE.

Elle contient un fichier de configuration appelé « application.xml ».



17

CHAPITRE II

ARCHITECTURE J2EE

1.2.2. La livraison Web WAR (Web ARchive)

Cette livraison contient :

- Des pages HTML, JSP
- Des images
- Des feuilles de styles (CSS, XSL, ...)
- Classes Java métier ou d'accès aux données
- Fichier de configuration « web.xml »

Ce livrable peut être livré seul dans un fichier .WAR ou à l'intérieur d'un EAR.

18

CHAPITRE II

ARCHITECTURE J2EE

1.2.3. La livraison EJB (Entreprise JavaBean) (.JAR)

Les EJB sont des composants java métiers évoluant dans un conteneur EJB d'un serveur d'application.

Ce livrable peut être livré seul dans un fichier .JAR ou à l'intérieur d'un EAR.

19

CHAPITRE II

ARCHITECTURE J2EE

1.2.4. La livraison Client (Module Client) (.JAR)

Ce livrable contient un client java pour interroger un EJB.

Ce livrable peut être livré seul dans un fichier .JAR ou à l'intérieur d'un EAR.

20

CHAPITRE II

ARCHITECTURE J2EE

1.2.5. La livraison Ressources (Module Connector) (.RAR)

Ce livrable .RAR (Ressources ARchive) fournit une solution pour le problème de **connectivité** entre plusieurs **serveurs d'applications** et un système d'information d'entreprise **EIS** existant.

Permet d'établir des communications synchrones du type demande / réponse avec un serveur. Il n'empêche pas d'utiliser un mode asynchrone et bidirectionnel.

Ce livrable peut être livré seul dans un fichier .RAR ou à l'intérieur d'un EAR.

21

CHAPITRE II

ARCHITECTURE J2EE

2. Vue d'ensemble d'application J2EE

2.1. Création

Dans notre cours, nous allons utiliser l'IDE **Eclipse**, et le serveur d'application open source **JBoss**.

La création d'un projet J2EE se fait après l'installation du serveur d'application JBoss et Eclipse.

Eclipse doit être configuré pour utilisé JBoss installé.

Eclipse permet de créer un projet J2EE par:

File/New/Entreprise Application Project

Nous allons voir les détails dans la séance du TP.

22

CHAPITRE II

ARCHITECTURE J2EE

2.2. Assemblage

Une application J2EE peut regrouper différents modules : modules web, modules EJB ...

J2EE propose de regrouper ces différents modules dans un module unique sous la forme d'un fichier **EAR** (Entreprise ARchive).

Un archive EAR contient:

- les modules qui composent l'application (web, EJB, RAR, ...)
- un fichier descripteur de déploiement « application.xml »

23

CHAPITRE II

ARCHITECTURE J2EE

2.2. Assemblage (suite)

Une application qui ne contient que le module web peut être assemblé dans un fichier WAR (Web Archive).

Eclipse permet de faire ces assemblage en utilisant la fonctionnalité « Export ».

Nous verrons les détails dans la séance de TP.

24

CHAPITRE II

ARCHITECTURE J2EE

2.3. Déploiement

Déployer un projet dans un serveur d'application, est de le placer dans le serveur pour pouvoir l'exécuter.

Le déploiement d'un projet peut se faire de deux manières:

- À partir d'Eclipse

En utilisant « Run As/Run on Server »

- À partir de la console d'administration de JBoss.

En utilisant « Manage Deployment/ Add Content »

Nous verrons plus de détails dans la séance de TP.

25

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

PLAN

1. Serveur EJB
2. Modèle de composants EJB

26

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

1. Serveur EJB

Un serveur EJB ou conteneur EJB fait partie de l'architecture d'un serveur d'application.

Il héberge les beans métier qui peuvent être appelés par l'utilisateur à travers la couche présentation (JSP, servlets).

27

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2. Modèle de composants EJB

2.1. C'est quoi un composant EJB

Un **composant EJB** contient la **partie logique** (le cœur) de l'application.

Les **clients** distants peuvent **appeler** les méthodes développées dans un EJB.

Le **conteneur** EJB fournit des services au niveau système, comme la gestion des **transactions** et la **sécurité** des composants EJB.

Le **développeur** des EJB s'occupe juste de la **logique** de l'application.

28

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2. Types de composants EJB

Il existe 3 types de composants EJB:

- Session (stateful, stateless)
- Entité (BMP, CMP)
- Message

29

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2. Types de composants EJB (suite)

2.2.1. EJB session

Un EJB session représente le **point d'entrée** d'un client.

Un EJB session ne peut recevoir **qu'un seul client** à un moment donné.

Le client appelle les méthodes de l'EJB session pour **accéder** à l'application.

Il existe 2 types d'EJB session:

- EJB session avec état (stateful);
- EJB session sans état (stateless).

30

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2.1. EJB session (suite)

L'état d'un composant (objet) EJB session est la valeur des variables d'instance de cet objet.

2.2.1.1. EJB session avec état (stateful)

Le client **interagit** (discute) avec son EJB session.

L'état de l'EJB est **maintenu** pour la durée de la session.

La session **termine** quand le processus **client termine**, ou le client **termine la session de l'EJB**.

31

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2.1. EJB session (suite)

2.2.1.2. EJB session sans état (stateless)

Dans ce cas, le client n'a **pas besoin** de conversation avec l'EJB session.

L'état de l'EJB session est **maintenu** pour la **durée** de la **méthode** appelée. Une fois la méthode termine, l'état de l'EJB est **perdu**, et un **autre client** peut utiliser cet EJB session.

32

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2.1. EJB session (suite)

2.2.1.3. Quand utiliser les composants EJB session

On les utilise dans les cas suivants:

- À un **moment donné**, un **seul** client doit accéder à l'instance de l'EJB;
- L'état de l'EJB n'est **pas persistant** (pas besoin d'être permanent). On a besoin de l'état de l'EJB pour une courte période;
- L'EJB implémente un **web service** (l'EJB offre un service et le client y accède à partir du web dont il a besoin d'une session).

33

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2.2. EJB entité

Un EJB **entité** représente un objet métier **persistant** (L'**état** de l'objet doit exister en permanence).

Exp. Un EJB entité représente un objet client, un objet commande ou un objet produit.

Un EJB entité peut être **utilisé** par **plusieurs clients** en même temps, puisque il existe en permanence.

34

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2.2. EJB entité (suite)

Il existe deux types de persistances pour l'EJB entité: persistance géré par le bean et persistance géré par le conteneur.

2.2.2.1. Persistance géré par le bean BMP (Bean Managed Persistence)

L'accès aux données persistantes (accès à la BDD) se fait dans l'EJB entité.

35

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2.2. EJB entité (suite)

2.2.2.1. Persistance géré par le conteneur CMP (Container Managed Persistence)

L'accès aux données persistantes (accès à la BDD) se fait par le conteneur, et l'EJB entité traite les opérations avancées.

Avantage: L'EJB entité CMP est plus **flexible** que l'EJB BMP, parce que si on **déploie** un l'EJB entité CMP dans un autre serveur d'application qui utilise une BDD différente, on n'aura **pas besoin** de le **modifier** et de le **recompiler**.

36

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.2.3. EJB message

Un EJB message permet aux applications J2EE de traiter des **messages** d'une manière **asynchrone**.

En général, par l'intermédiaire de JMS (Java Message Service).

37

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3. Les interfaces EJB

Les interfaces EJB sont définies pour les EJB **entité** et **session**.

Un client ne peut accéder à un EJB que par l'**intermédiaire** d'une interface.

Il existe 2 types d'interfaces: **Interface Home** (Home interface) et **Interface métier** (Component interface)

L'interface Home contient les définitions des méthodes de **création**, **suppression** ou de **recherche** d'un bean.

L'interface métier contient les définitions des **méthodes métiers** contenues dans le bean.

38

CHAPITRE III

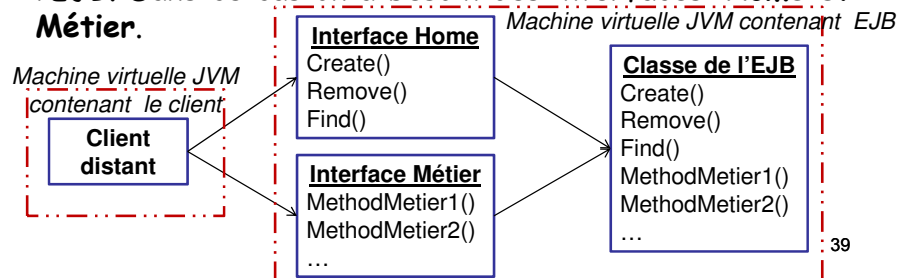
COUCHE MÉTIER ET PERSISTANCE

2.3. Les interfaces EJB (suite)

Un client peut accéder à un EJB de 3 manière différentes: distant, local et service web.

2.3.1. Accès distant

Le client tourne sur une JVM différente de celle de l'EJB. Dans ce cas on a besoin des interfaces: **Home** et **Métier**.



39

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3.1. Accès distant (suite)

Exp: Interface Home

```
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

public interface CartHome extends EJBHome {
    Cart create(String person) throws RemoteException,
        CreateException;
    Cart create(String person, String id) throws
        RemoteException, CreateException;
}
```

40

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3.1. Accès distant (suite)

Exp: Interface métier

```
import java.util.*;
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
public interface Cart extends EJBObject {
    public void addBook(String title) throws
        RemoteException;
    public void removeBook(String title) throws
        BookException, RemoteException;
    public Vector getContents() throws RemoteException;
}
```

41

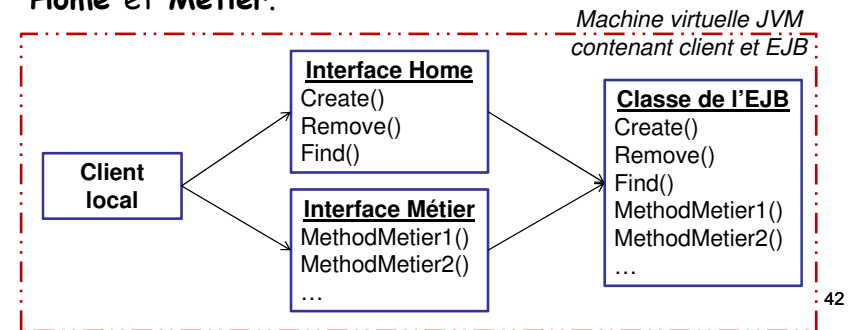
CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3. Les interfaces EJB (suite)

2.3.2. Accès local

Le client tourne sur la même machine virtuelle JVM que celle de l'EJB. Dans ce cas on a besoin des interfaces: **Home** et **Métier**.



42

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3.2. Accès local (suite)

Exp: Interface métier

```
package team;
import java.util.*;
import javax.ejb.*;
public interface LocalPlayer extends EJBLocalObject {
    public String getPlayerId();
    public String getName();
    public String getPosition();
    public double getSalary();
    public Collection getTeams();
    public Collection getLeagues() throws FinderException;
    public Collection getSports() throws FinderException;
}
```

43

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3.2. Accès local (suite)

Exp (suite): Interface Home

```
package team;
import java.util.*;
import javax.ejb.*;
public interface LocalPlayerHome extends EJBLocalHome {
    public LocalPlayer create (String id, String name,
        String position, double salary) throws CreateException;
    public LocalPlayer findByPrimaryKey (String id) throws
        FinderException;
    public Collection findByPosition(String position) throws
        FinderException;
}
```

44

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3. Les interfaces EJB (suite)

2.3.3. Accès par service Web

Un client web service peut accéder à l'application J2EE soit par un service web créé par JAX-RPC ou bien en appelant les méthodes d'un **EJB session sans état**.

Dans ce cas, nous n'avons besoin que d'une **interface métier**, puisque le client n'est pas responsable de la création et suppression du service web.

45

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3. Les interfaces EJB (suite)

2.3.3. Accès par service Web (suite)

Exp: Interface métier

```
package helloservice;
import java.rmi.RemoteException;
import java.rmi.Remote;
public interface HelloService extends Remote {
    public String sayHello(String name) throws
        RemoteException;
}
```

46

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3. Les interfaces EJB (suite)

2.3.3. Accès par service Web (suite)

Exp (suite): Classe de l'EJB session sans état

```
package helloservice;
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
public class HelloServiceBean implements SessionBean {
    public String sayHello(String name) {
        return "Hello " + name + " from HelloServiceBean";
    }
    public HelloServiceBean() {}
}
```

47

CHAPITRE III

COUCHE MÉTIER ET PERSISTANCE

2.3. Les interfaces EJB (suite)

Remarque:

Au contraire des EJB session et entité, les EJB message (Message-Driven Beans) n'ont pas besoin d'interfaces Home et métier pour définir le type d'accès des clients (local ou distant).

Le client n'a pas besoin de localiser l'EJB message, puisque la communication est asynchrone.

Un tel EJB n'a pas de méthodes métier. La méthode importante est `onMessage()`. Qui est appelée à la réception d'un message.

48

CHAPITRE IV

COUCHE PRÉSENTATION

PLAN

1. Les servlets
2. La technologie JSP
3. Communication entre servlet et vue
4. La session en J2EE

49

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets

Une servlet est une classe java capable de recevoir une requête HTTP envoyée depuis le navigateur de l'utilisateur, et de lui renvoyer une réponse HTTP.

Exp:

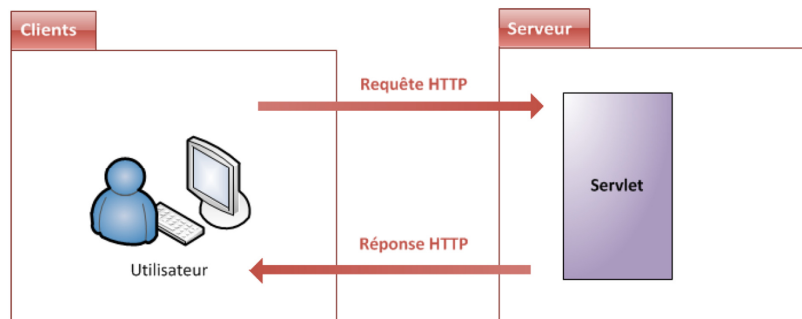
```
public class Test extends HttpServlet {  
    // Corps de la servlet  
}
```

50

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)



51

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)

La classe de la servlet doit définir les méthodes héritées suivantes selon le besoin:

- `doGet()` pour gérer la méthode GET (Méthode utilisée par le client pour récupérer une ressource web du serveur via une URL);
- `doPost()` pour gérer la méthode POST (Méthode utilisée pour envoyer des informations au serveur);
- `doHead()` pour gérer la méthode HEAD (Même travail que la méthode GET sauf que l'utilisateur récupère seulement l'entête HTTP).

52

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)

Il faut associer la servlet à une URL. Quand le client saisira l'URL, la requête HTTP sera automatiquement aiguillée par le conteneur de servlet vers la bonne servlet.

Cette association se fait dans le fichier « **web.xml** » de l'application J2EE.

53

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)

Exp:

```
<servlet>
  <servlet-name>Test</servlet-name>
  <servlet-class>package.class</servlet-class>
  <description>Description de servlet</description>
  <init-param>
    <param-name>param1</param-name>
    <param-value>val1</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

54

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)

Tel que:

`<description>` permet de décrire le rôle de la servlet.

`<init-param>` permet de préciser des paramètres qui seront accessibles à la servlet lors de son chargement.

`<load-on-startup>` permet de forcer le chargement de la servlet dès le démarrage du serveur. Dans le cas d'une application contenant plusieurs servlets, cela permet de définir quelle servlet doit être chargée en premier (0 avant 1 avant 2 ...).

55

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)

Exp: (suite)

```
<servlet-mapping>
  <servlet-name>Test</servlet-name>
  <url-pattern>/toto</url-pattern>
</servlet-mapping>
```

Il est possible de déclarer plusieurs sections `<servlet-mapping>` pour une même section `<servlet>` dans le fichier « **web.xml** » dans le but d'associer plusieurs URL à une servlet.

56

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)

Nous devons maintenant surcharger la méthode `doGet()` de la classe `HttpServlet`.

Exp:

```
public class Test extends HttpServlet {
    public void doGet ( HttpServletRequest request ,
        HttpServletResponse response ) throws
        ServletException , IOException {
        //Corps de la méthode
    }
}
```

57

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)

Reste à générer la réponse à envoyer à l'utilisateur. Cela se fait dans l'objet `HttpServletResponse` comme suit:

```
response.setContentType("text/html");
response.setCharacterEncoding("UTF-8");
PrintWriter out = response.getWriter();
out.println("<!DOCTYPE html>");
out.println("<html>");
out.println("<head>");
out.println("<meta charset =\"utf-8\" />");
out.println("<title>Test</title>");
out.println("</head>");
out.println("<body>");
out.println("<p>Page générée depuis une servlet.</p>");
out.println("</body>");
out.println("</html>");
```

58

CHAPITRE IV

COUCHE PRÉSENTATION

1. Les servlets (suite)

Maintenant, il faut déployer le projet J2EE et appeler la servlet:

`http://localhost:8080/Capitre4_ServletWeb/toto`

Tel que: « `Chapitre4_ServletWeb` » est le module Web du projet J2EE « `Chapitre4_Servlet` »

59

CHAPITRE IV

COUCHE PRÉSENTATION

2. La technologie JSP

L'intérêt est de rendre possible la création de pages **dynamiques**. Il est possible de varier l'affichage et d'interagir avec l'utilisateur.

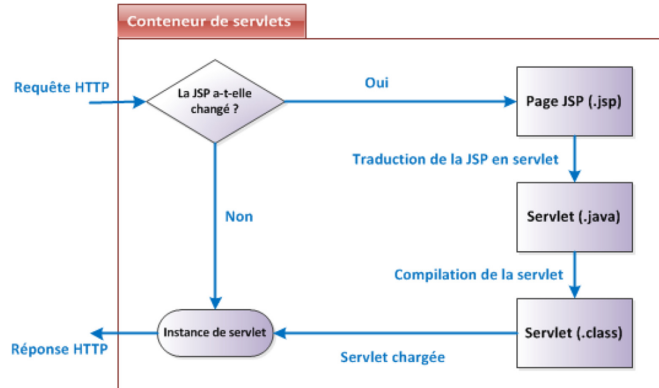
60

CHAPITRE IV

COUCHE PRÉSENTATION

2. La technologie JSP (suite)

Cycle de vie d'une JSP:



61

CHAPITRE IV

COUCHE PRÉSENTATION

2. La technologie JSP (suite)

Comme le montre le schéma précédent, une page JSP est traduite en une servlet.

Une bonne pratique est d'associer une **servlet de contrôle** à une vue (la page JSP).

Remarque:

Il ne faut pas confondre entre une servlet de contrôle et la servlet générée à partir de la page JSP.

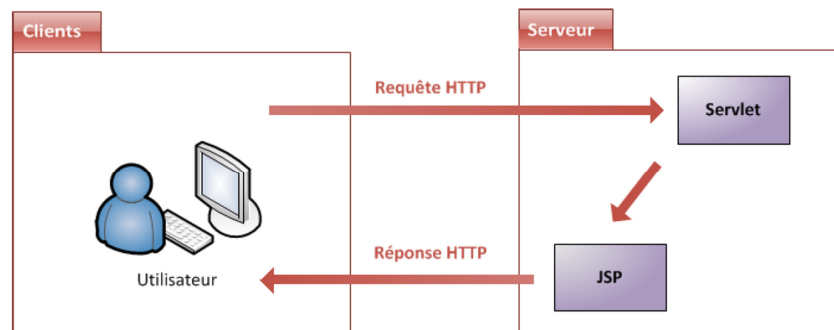
62

CHAPITRE IV

COUCHE PRÉSENTATION

2. La technologie JSP (suite)

Vue avec servlet de contrôle:



63

CHAPITRE IV

COUCHE PRÉSENTATION

2. La technologie JSP (suite)

Exp: Vue appelée « test.jsp »:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
  <meta charset=utf-8">
  <title>Page de teste</title>
</head>
<body>
  <p>Exemple d'une servlet avec vue (Page JSP)... </p>
</body>
</html>
```

64

CHAPITRE IV

COUCHE PRÉSENTATION

2. La technologie JSP (suite)

Pour associer une servlet de contrôle à la page JSP nous devons placer le code suivant dans la méthode `doGet()` de la servlet:

```
public void doGet (HttpServletRequest request,
    HttpServletResponse response) throws
    ServletException, IOException {
    this.getServletContext().getRequestDispatcher("/
    WEB-INF/test.jsp").forward(request, response);
}
```

Tel que la page « test.jsp » est placée dans le répertoire « WEB-INF » ainsi que le fichier « web.xml »

65

CHAPITRE IV

COUCHE PRÉSENTATION

3. Communication entre servlet et vue

3.1. Passage de variables de la servlet à la vue

Pour transmettre une variable à la vue, on utilise la méthode `setAttribute()` de l'objet `HttpServletRequest` dans la méthode `doGet()` comme suit:

Exp:

```
String message = "Exemple de passage de variables de
la servlet à la vue...";
request.setAttribute("var1", message);
this.getServletContext().getRequestDispatcher("/WEB-
INF/test.jsp").forward(request, response);
```

66

CHAPITRE IV

COUCHE PRÉSENTATION

3.1. Passage de variables de la servlet à la vue (suite)

La méthode `setAttribute()` prend en arguments le nom de la variable et la valeur.

Pour récupérer la variable dans la vue on utilise la méthode `getAttribute()` de l'objet `HttpServletRequest`.

67

CHAPITRE IV

COUCHE PRÉSENTATION

3.1. Passage de variables de la servlet à la vue (suite)

Exp:

Dans la page « test.jsp » de l'exemple précédent:

```
<p>Exemple d'une servlet avec vue (Page JSP)...</p>
<p>
    <%
        String attribut = (String) request.getAttribute("var1");
        out.println(attribut);
    %>
</p>
```

68

CHAPITRE IV

COUCHE PRÉSENTATION

3.2. Passage de variables du client au serveur

L'utilisateur peut envoyer des données sous forme de paramètres dans l'URL.

Exp:

```
http://localhost:8080/Chapitre4_Servlet+JSPWeb/toto?param1=Bonjour&param2=true&param3=5
```

Le type des variables passées en paramètre est: **String**.

69

CHAPITRE IV

COUCHE PRÉSENTATION

3.2. Passage de variables du client au serveur (suite)

Le serveur récupère les informations en utilisant la méthode `getParameter()` de l'objet `HttpServletRequest` dans la méthode `doGet()`.

Exp: Dans l'exemple précédent:

```
String paramUser = "Val1 =  
    "+request.getParameter("param1")+" - Val2 =  
    "+request.getParameter("param2")+" - Val3 =  
    "+request.getParameter("param3");  
String message2 = "Exemple de passage de variables  
    du client au serveur "+paramUser;  
request.setAttribute("var2", message2);  
this.getServletContext().getRequestDispatcher("/WEB-INF/test.jsp").forward(request, response);
```

70

CHAPITRE IV

COUCHE PRÉSENTATION

3.2. Passage de variables du client au serveur (suite)

On peut récupérer les paramètres de l'URL à partir de la page JSP directement:

Exp:

```
<p>  
    <%  
        String paramUser = "Val1 =  
        "+request.getParameter("param1")+" - Val2 =  
        "+request.getParameter("param2")+" - Val3 =  
        "+request.getParameter("param3");  
        out.println(paramUser);  
    %>  
</p>
```

71

CHAPITRE IV

COUCHE PRÉSENTATION

3.3. Passage de données par formulaire

Exp:

Nous allons réaliser une simple page web contenant un formulaire destiné à l'inscription du visiteur sur un site. Ce formulaire contient :

- un champ texte pour l'adresse email de l'utilisateur ;
- un champ texte pour le mot de passe ;
- un champ texte pour la confirmation du mot de passe ;
- un champ texte pour le nom d'utilisateur.

72

CHAPITRE IV

COUCHE PRÉSENTATION

3.3. Passage de données par formulaire (suite)

Étapes à suivre:

1. Création d'un bean qui enregistre les données saisies;
2. Création d'un objet métier comportant les méthodes de récupération/conversion/validation des contenus des champs du formulaire;
3. Création d'une servlet de contrôle pour aiguiller la requête entrante;
4. Création de la vue (page JSP).

73

CHAPITRE IV

COUCHE PRÉSENTATION

3.3. Passage de données par formulaire (suite)

a) Création d'un bean qui enregistre les données saisies:

```
public class Utilisateur {  
    private String email;  
    private String name;  
    private String pass;  
    ... //getters et setters  
}
```

74

CHAPITRE IV

COUCHE PRÉSENTATION

3.3. Passage de données par formulaire (suite)

b) Création d'un objet métier comportant les méthodes de récupération/conversion/validation des contenus des champs du formulaire:

```
public class InscriptionForm {  
    ... //suite inscrireUtilisateur()  
}
```

75

CHAPITRE IV

COUCHE PRÉSENTATION

3.3. Passage de données par formulaire (suite)

```
public class InscriptionForm {  
    public Utilisateur inscrireUtilisateur(HttpServletRequest request){  
        String email = request.getParameter("email");  
        String motDePasse = request.getParameter("motdepasse");  
        String confirmation = request.getParameter("confirmation");  
        String nom = request.getParameter("nom");  
        Utilisateur utilisateur = null;  
        if (motDePasse.equals(confirmation)) {  
            utilisateur = new Utilisateur();  
            utilisateur.setEmail(email);  
            utilisateur.setPass(motDePasse);  
            utilisateur.setName(nom);  
        }  
        return utilisateur;  
    }  
}
```

76

CHAPITRE IV

COUCHE PRÉSENTATION

3.3. Passage de données par formulaire (suite)

c) Création d'une servlet de contrôle pour aiguiller la requête entrante:

```
public class Inscription extends HttpServlet {
    public static final String VUE = "/WEB-INF/inscription.jsp";
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
        this.getServletContext().getRequestDispatcher(VUE).forward(
            request, response);
    }
}
```

77

CHAPITRE IV

COUCHE PRÉSENTATION

3.3. Passage de données par formulaire (suite)

c) Création d'une servlet de contrôle pour aiguiller la requête entrante: (suite)

```
protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
    InscriptionForm form = new InscriptionForm();
    Utilisateur user = form.inscrireUtilisateur(request);
    request.setAttribute("utilisateur", user);
    this.getServletContext().getRequestDispatcher(VUE).forward(
        request, response);
}
}
```

78

CHAPITRE IV

COUCHE PRÉSENTATION

3.3. Passage de données par formulaire (suite)

d) Création de la vue (page JSP):

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="UTF-8"%>
<%@ page import="org.Utilisateur" %>
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Inscription</title>
</head>
<body>
<%
Utilisateur user =(Utilisateur)request.getAttribute("utilisateur");
if (user != null) {
%>
```

79

CHAPITRE IV

COUCHE PRÉSENTATION

d) Création de la vue (page JSP): (suite)

```
<p>Bonjour <%out.print(user.getName());%><br>
Votre email : <%out.print(user.getEmail()); %><br>
Votre mot de passe : <%out.print(user.getPass()); %><br> </p>
<%> else {%>
<form method="post" action="toto">
<fieldset>
<legend>Inscription</legend>
<p>Vous pouvez vous inscrire via ce formulaire .</p>
<label for="email">Adresse email <span class="requis">*</span>
</label>
<input type="email" id="email" name="email" size="20"
maxlength="60"/><br/>
<label for="motdepasse">Mot de passe<span
class="requis">*</span></label>
```

80

CHAPITRE IV

COUCHE PRÉSENTATION

d) Création de la vue (page JSP): (suite)

```
<input type="password" id="motdepasse" name="motdepasse" value=""
size="20" maxlength="20"/><br/>
<label for="confirmation">Confirmation du mot de passe<span
class="requis">*</span></label>
<input type="password" id="confirmation" name="confirmation"
value="" size="20" maxlength="20"/><br/>
<label for="nom ">Nom d'utilisateur</label>
<input type="text" id="nom" name="nom" size="20" maxlength
="20"/><br/>
<input type="submit" value="Inscription"/><br/>
</fieldset>
</form>
<%}%>
</body>
</html>
```

81

CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE

La session est un ensemble d'informations d'un utilisateur stocké dans un espace mémoire;

Le contenu de la session est conservé jusqu'à ce que l'utilisateur **ferme son navigateur**, reste **inactif** trop longtemps, ou se **déconnecte** du site.

82

CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE (suite)

Pour créer une session nous avons besoin d'une servlet.

La méthode `getSession()` de l'objet `HttpServletRequest` permet de récupérer la session (de type `HttpSession`) associée à la requête HTTP si elle existe, sinon elle en crée une:

Exp:

```
HttpSession session = request.getSession();
```

83

CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE (suite)

Le classe `HttpSession` possède les méthodes `setAttribute()`, `getAttribute()` comme nous les avons déjà vue et `getId()` pour récupérer l'identifiant de la session.

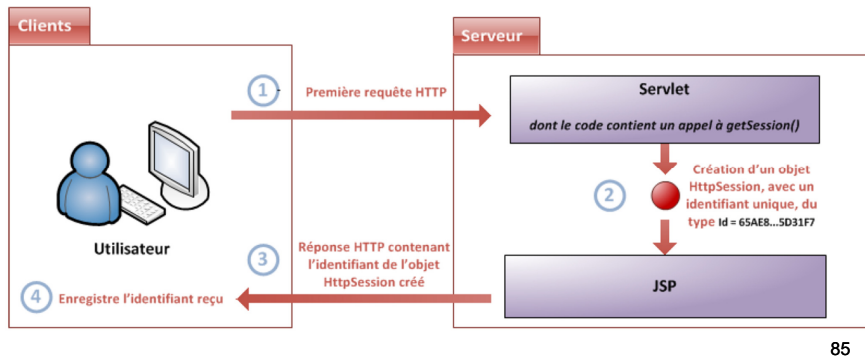
84

CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE (suite)

Enchaînement des opérations de la 1^{ère} connexion:

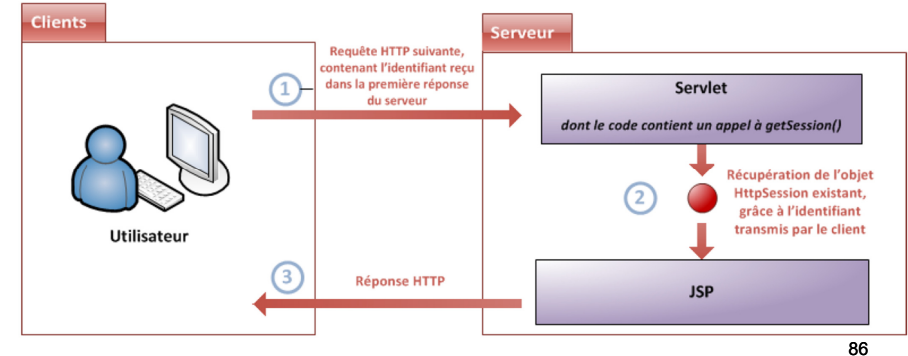


CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE (suite)

Enchaînement des opérations des connexions suivantes:



CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE (suite)

Pour utiliser une session, dans `doPost()` nous avons besoin de:

- Récupérer l'objet session à partir de la requête:

```
HttpSession session = request.getSession();
```
- On peut envoyer des informations à la vue avec `setAttribute()`:

```
session.setAttribute("var1", InfoUser);
```

CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE (suite)

et dans la vue JSP nous avons besoin de récupérer les informations envoyées par la servlet:

- Soit en utilisant `session.getAttribute("var1")`
- Soit en utilisant l'objet implicite `sessionScope`. C'est un objet dédié à l'accès au contenu de la session:

```
${sessionScope.var1}
```

CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE (suite)

Pour la **déconnexion**:

- Soit l'utilisateur ferme le navigateur;
- Soit la session expire après une période d'inactivité;

La période est définie en **minutes** dans « web.xml » comme suit:

```
<session-config>
  <session-timeout>1</session-timeout>
</session-config>
```

- Soit l'utilisateur se déconnecte.

CHAPITRE IV

COUCHE PRÉSENTATION

3.4. La session en J2EE (suite)

Dans le cas où l'utilisateur se déconnecte, nous avons besoin d'une autre **servlet**. Dans sa méthode `doGet()` on récupère l'objet session:

```
HttpSession session = request.getSession();
```

Puis on fait appel à la méthode:

```
session.invalidate();
```

Cette méthode **supprime** la session et les objets qu'elle contient, et envoie une exception si la session est déjà détruite.